Remarks

Introductory Comments

Prior to this amendment, the present application included claims 67-100. With this amendment, Applicants cancel claims 73-77, without prejudice or disclaimer of the subject matter thereof, and amend claim 95. This application now includes claims 67-72 and 78-100. Applicants respectfully request reconsideration of the application in view of the following remarks, and that the Examiner find claims 67-72 and 78-100 in condition for allowance.

Claim Rejections – 35 USC § 112, first paragraph

Claims 67-87 were rejected under 35 USC § 112, first paragraph, as failing to comply with the written description requirement. In particular, the Examiner lists the steps of independent claims 67, 73, 78 and 82 and states "none of these methods to add a silent guard are disclosed in the Specification." (Office Action, page 3, lines 20-21) The other claims in the range 67-87 depend on these four independent claims. In this amendment, Applicants cancel claim 73 and the claims dependent thereon and respectfully traverse Examiner's rejection regarding claims 67-72 and 78-87.

The subject matter covered in claims 67-72 and 78-87 is covered in several places throughout the specification, including page 28, line 4 – page 34, line 10 and page 84, line 22 – page 88, line 6. Specific references to some of the occurrences of description in the specification supporting the subject matter of independent claims 67, 78 and 82 is detailed below. These references are given by way of example and are not intended to limit the interpretation of the claims. In fact, it will be appreciated by one of ordinary skill in the art that there are many other ways of implementing the steps recited in the claims.

Claim 67

Claim 67 is directed to a computer implemented method for adding tamper resistance to a software program. This method is outlined in the specification, and an example and explanation of one implementation is given on page 31, line 15 – page 34, line 10.

Description supporting the step of "adding a silent guard variable to the software program" can be found on page 28, lines 17-19 which read "the variable selected for silent guarding may be a specialized variable installed in the application software program for the purpose of the silent guard;" and also in the example on page 31, lines 18-19 which read "the silent guard comprises a variable X . . . Also shown is a variable X1 whose value is computed during the computation of variable X . . . This example also comprises a flag variable F." In the example, silent guard variables X, X1 and F are added to the software program.

Description supporting the step of "selecting a computation in the software program" can be found in the example on page 32 where line 8 has the comment "sample computation of V" and on page 31, lines 16-17 which read "In the pseudocode segment of the following example, variable V is a program variable used in the application software program." In the example, the computation of variable V in the software program is selected for silent guarding. This is also further supported by the description referenced below in support of the following steps.

Description supporting the step of "determining an expected value of the silent guard variable at the execution point of the selected computation" can be found at page 29, lines 3-5 which read "the variable selected for silent guarding has an expected value, i.e., the value the variable should have at a particular point in program execution" and in the example on page 31, lines 18-22 which read:

> the silent guard comprises a variable X whose expected value is 'TRUE.' Also shown is a second variable X1 whose value is computed during the computation of variable X. Variable X1 has an expected value T1. ... The

value of flag variable F is initialized to 0, and is changed to 1 after the silent guard program instructions are executed.

In the example, each of the silent guard variables has an expected value, the expected value of the silent guard variable X1 is determined to be T1 at the execution point of the selected computation.

Description supporting the step of "revising the selected computation to be dependent on the runtime value of the silent guard variable, such that the selected computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable" can be found on page 33, line 21 – page 34, line 10, which reads:

> The third protection contained in the foregoing pseudocode example arises from the fact that the value of program variable V computed at line 10 will be correct only if the value of Y1 is 0. Thus, the pseudocode program instruction "Y1=X1-T1" shown at line 9 comprises a conditional computation according to the present invention. <u>The value of Y1 will be computed correctly only if the value of variable X1 computed at line 2 is the same as its expected value T1. Accordingly, if the client code block of variable X1 is altered unexpectedly, such as by the actions of a hacker, the value of computed variable X1 will not be the same as its expected value T1. If X1 $\neq$ T1, the value of variable Y1 computed at line 9 will be inaccurate and, thus, program variable V will be inaccurately computed at line 10.</u> As before, depending on where and how program variable V is used in the application software program, the inaccurately computed value of program variable V may not become manifest until later in program execution. <u>A hacker altering the computation of variable X1 may have difficulty connecting the altered computation of variable X1 with the failure caused by the inaccurately computed program variable V.</u> (emphasis added)

Therefore, since each of the steps of claim 67 is described in the Specification along with an example of their implementation, Applicants respectfully request that this rejection of claim 67 be withdrawn. Claims 68-72 are dependent on claim 67 and are also described in the specification. Applicants respectfully request that this rejection of claims 68-72 also be withdrawn.

Claim 78

Claim 78 is directed to a computer implemented method for adding tamper resistance to a software program. This method is outlined in the specification, and an example and explanation of the implementation is given on page 31, line 15 – page 34, line 10 and on page 84, line 22 – page 86, line 7.

Description supporting the step of "selecting a program variable in the software program" can be found on page 28, lines 16-17 which read "Typically, the variable selected for silent guarding is a program variable used by the application software program to carry out its intended function;" and lines 21-23 which read "For example, a variable whose value comprises a password entered by a user of the application software program may be selected for silent guarding." This is also shown in the example on page 32 of the specification, in which the variable "V" (pseudocode lines 8, 10) "is a program variable used in the application software program." (page 31, line 17). This is also shown in the example on page 85 of the specification, in which the variable "*p*" is a password selected for silent guarding (page 85, lines 1-2).

The step of "selecting a computation in the software program" is shown in the example on page 32 where line 8 has the comment "sample computation of V" and on page 31, lines 16-17 which read "In the pseudocode segment of the following example, variable V is a program variable used in the application software program." In the example, the computation of variable V in the software program is selected for silent guarding. In the example on page 85, the computation of variable x, "where x is a program variable used in the application software program" (page 85, line 21) is selected. The statement that "x is a program variable used in the application software program" clearly discloses that x is used in "a computation in the software program," especially in view of the example on page 32. This is also further supported by the description cited below supporting the following steps.

Description supporting the step of "determining an expected value of the program variable at the point of execution of the selected computation" is shown in the example on page 85 which states "password p whose value should be 81" (page 85, line 1). The expected value of program variable p is 81. Description supporting the step of "revising the selected computation to be dependent on the runtime value of the program variable, such that the selected computation will evaluate incorrectly if the runtime value of the program variable is not equal to the expected value of the program variable" is also shown in the example on page 85 which on lines 18-21 shows the sequence of steps:

$$t = \sqrt{p}$$

$$v = (t + 1)^2 - t^2 - 18$$

$$x = x * v$$

and the following explanation "if $p = 81$, then $v = 1$ and the value of program variable x is computed correctly. However, if $p \neq 81$, then $v \neq 1$. If $v \neq 1$, the value of program variable x is computed incorrectly, and the program is corrupted." (page 85, line 21-page 86, line 2). Thus, the computation of x and any calculation in the application program using x, has effectively been revised to be dependent on the runtime value of the program variable p, such that the computation using x will evaluate incorrectly if the runtime value of the program variable p is not equal to the expected value of the program variable p.

Therefore, since each of the steps of claim 78 is described in the Specification along with an example of their implementation, Applicants respectfully request that this rejection of claim 78 be withdrawn. Claims 79-81 are dependent on claim 78 and are also described in the specification. Applicants respectfully request that this rejection of claims 79-81 also be withdrawn.

Claim 82

Claim 82 is directed to a computer implemented method for adding tamper resistance to a software program. This method is outlined in the specification, and an example and explanation of one implementation is given on page 31, line 15 – page 32, line 20.

In the example referenced, the step of "selecting a program block containing a step necessary for proper execution of the software program, the program block comprising at least one program instruction" is shown on page 32, lines 18-20 which is a program block of three code lines:

```
8          V = Z * (A*Z + B) + 72
9          Y1 = X1-T1
10         V = V+Y1
```

At least the computation of V at program line 8 being necessary for proper program execution (page 33, lines 13-15). The step of "selecting a silent guard for the program block" is shown by selecting the variable F which "acts as a silent guard." (page 33, line 4). The step of "determining the expected value of the silent guard at the start of execution of the program block" is shown by the expected value of F being 1 (see page 31, lines 20-22; page 33, lines 4-6). The step of "installing a branch instruction dependent on the silent guard in the software program, such that if the runtime value of the silent guard is not equal to the expected value of the silent guard then the branch instruction will cause an incorrect branch to be taken" is shown by the instruction on page

32 at pseudocode line 7 which is dependent on the silent guard F and which will cause an incorrect branch to L1 (pseudocode line 11) if the silent guard F is not equal to its expected value of 1 (see explanation on page 33, lines 4-20).

Therefore, since each of the steps of claim 82 is described in the Specification along with an example of their implementation, Applicants respectfully request that this rejection of claim 82 be withdrawn. Claims 83-87 are dependent on claim 82 and are also described in the specification. Applicants respectfully request that this rejection of claims 83-87 also be withdrawn.

Claims 70 and 79

Claims 70 and 79 were also rejected under 35 USC § 112, first paragraph, as failing to comply with the written description requirement.

The subject matter covered in claims 70 and 79 is covered in several places throughout the specification, especially in discussions of conditional computations and conditional identities. The following references are given by way of example and are not intended to limit the interpretation of the claims. In fact, it will be appreciated by one of ordinary skill in the art that there are many other ways of implementing the steps recited in the claims.

Claim 70

Claim 70 is dependent on claim 67 and includes some additional steps for revising the selected computation. The example on page 85 which has been discussed above, also exhibits the steps of claim 70. Claim 70 recites "selecting a constant value used in the selected computation." The selected computation in the example on page 85 is a computation using x which "is a program variable used in the application software program." (page 85, line 21) It is well known in the art, and in mathematics in general, that any variable z can be replaced by z*1 due to the multiplicative identity principle, and the number "1" is a constant. In fact, the expression on page 85, line 20 is effectively $x = x * 1$, when v is equal to its expected value of "1."

The steps on lines 19-20 of page 85 show "replacing the constant value with a mathematical expression that is dependent on the runtime value of the silent guard variable, such that the mathematical expression evaluates to the constant value if the runtime value of the silent guard variable is equal to the expected value of the silent guard variable."

In line 20, the constant value "1" is replaced by the variable v, and line 19 shows the "mathematical expression that is dependent on the runtime value of the silent guard variable," the silent guard variable being t. It is obvious that the variable v in the expression on line 20 could be replaced by the mathematical expression for v shown in line 19 resulting in:

$$x = x * [(t + 1)^2 - t^2 - 18]$$

which is effectively replacing the constant value "1" with "a mathematical expression that is dependent on the runtime value of the silent guard variable" t as recited in claim 70.

As stated on page 85, line 21-page 86, line 2, "Note that if $p = 81$, then $v = 1$ and the value of program variable x is computed correctly. However, if $p \neq 81$, then $v \neq 1$. If $v \neq 1$, the value of program variable x is computed incorrectly, and the program is corrupted." The silent guard variable t which is used in the mathematical expression on line 19, is computed on line 18 as "$t = \sqrt{p}$." The expected value of p is 81 (page 85, line 1), and "the expected value of variable t is computed as $t = \sqrt{81} = 9$" (page 85, line 3). Thus, the statement on page 85, line 21-page 86, line 2 could be rephrased as "if $t = 9$, then $v = 1$ and the value of program variable x is computed correctly. However, if $t \neq 9$, then $v \neq 1$ and the value of program variable x is computed incorrectly, and the program is corrupted." Therefore, as recited in claim 70, "the mathematical expression evaluates to the constant value [1] if the runtime value of the silent guard variable [t] is equal to the expected value [9] of the silent guard variable."

Therefore, since each of the steps of claim 70 is described in the Specification along with an example of their implementation, Applicants respectfully request that this rejection be withdrawn.

Claim 79

Claim 79 is dependent on claim 78 and includes some additional steps for revising the selected computation. The additional steps of claim 79 are similar to the additional steps of claim 70 except that the mathematical expression of claim 79 is dependent on a program variable, whereas the mathematical expression of claim 70 is dependent on a silent guard variable. The example on page 85 also exhibits the additional steps of claim 79.

Claim 79 recites "selecting a constant value used in the selected computation." The selected computation in the example on page 85 is a computation using x which "is a program variable used in the application software program." (page 85, line 21) It is well known in the art, and in mathematics in general, that any variable z can be replaced by $z*1$ due to the multiplicative identity principle, and the number "1" is a constant. In fact, the expression on page 85, line 20 is effectively $x = x * 1$, when v is equal to its expected value of "1."

The steps on lines 19-20 of page 85 show "replacing the constant value with a mathematical expression that is dependent on the runtime value of the program variable, such that the mathematical expression evaluates to the constant value if the runtime value of the program variable is equal to the expected value of the program variable."

In line 20, the constant value "1" is replaced by the variable v, and lines 18 and 19 show the "mathematical expression that is dependent on the runtime value of the program variable," the program variable being p. It is obvious that the variable v in the expression on line 20 could be replaced by the mathematical expression for v shown in line 19 resulting in:

$$x = x * [(t + 1)^2 - t^2 - 18].$$

As shown on line 18 ($t = \sqrt{p}$), t is dependent on the program variable p which makes the mathematical expression dependent on the program variable p. This is effectively replacing the constant value "1" with "a mathematical expression that is dependent on the runtime value of the program variable" p as recited in claim 79.

As stated on page 85, line 21-page 86, line 2, "Note that if $p = 81$, then $v = 1$ and the value of program variable x is computed correctly. However, if $p \neq 81$, then $v \neq 1$. If $v \neq 1$, the value of program variable x is computed incorrectly, and the program is corrupted." Thus, as recited in claim 79, "the mathematical expression evaluates to the constant value [1] if the runtime value of the program variable [p] is equal to the expected value [81] of the program variable."

Therefore, since each of the steps of claim 79 is described in the Specification along with an example of their implementation, Applicants respectfully request that this rejection be withdrawn.

Claim Rejections – 35 USC § 112, second paragraph

Claim 95 was rejected under 35 USC § 112, second paragraph, for having insufficient antecedent basis for the limitation "the second dependency point." Claim 95 has been amended to recite the necessary antecedent basis for this limitation. Applicants respectfully request that this rejection be withdrawn.

Claim Rejections – 35 USC § 102(a)

Claims 67-69, 71-78 and 80-100 were rejected under 35 USC § 102(a) as being anticipated by the Collberg article entitled "A Taxonomy of Obfuscation Transformations" (hereinafter "Collberg"). As mentioned above, by this amendment, claims 73-77 are cancelled.

Thus, the following section addresses claims 67-69, 71, 72, 78 and 80-100 of the present application.

Collberg identifies 4 classes of technical program protection: *Obfuscation, Encryption, Server side execution* and *Trusted code* (Figure 1(a)), and then focuses almost entirely on obfuscation (Figure 1(b)-(g); Abstract, page 1, col. 1; and Introduction, page 2, col. 1, line 4 - col. 2, line 7). Collberg divides the targets of obfuscating transformations into 4 categories, *Layout obfuscation*, *Data obfuscation*, *Control obfuscation* and *Preventive transformation* (Figure 1(c)) and then presents a catalogue of obfuscating transformations for each of these categories (Figure 1(d)-(g)).

The Examiner rejects claims 67-69, 71, 72, 78 and 80-100 based on Collberg, Section 7.1.3 entitled "Split Variable" (page 18) and the accompanying Figure 18 (page 19). Split variable is a type of data obfuscation transformation (Collberg, page 2, Figure 1(e); page 17, Section 7 entitled "Data Transformations"). The data obfuscation transformations described in Section 7.1.3 of Collberg are one example of obfuscating transformations which Collberg defines on page 6, column 1, as:

> DEFINITION 1: (OBFUSCATING TRANSFORMATION)
>
> Let P $\xrightarrow{T}$ P' be a transformation of a source program P into a target program P'.
>
> P $\xrightarrow{T}$ P' is an *obfuscating transformation*, if P and P' have the same observable behavior. More precisely, in order for P $\xrightarrow{T}$ P' to be a legal obfuscating transformation the following conditions must hold:
>
> - If P fails to terminate or terminates with an error condition, then P' may or may not terminate.

- Otherwise, P' must terminate and produce the same output as P.

Thus, all of the obfuscation transformations in Collberg, including the split variable data transformation technique described in Section 7.1.3, must fit this definition of obfuscation transformation. In effect, Collberg requires that: If the original program P terminates without an error condition, then the new target program P' must terminate and produce the same output as P.

The spreadsheet attached to this response shows that for all of the example obfuscation transformations shown in Figure 18(e) of Collberg, the instructions before transformation, lines (5)-(10), produce the same outputs as the instructions after transformation, lines (5')-(10'), in accordance with the above definition of obfuscating transformation. For example, the output of lines (5)-(7) is C which has the same set of values before (spreadsheet col. 3) and after (spreadsheet col. 15) transformation regardless of the values of A and B or how A and B are transformed to a1, a2, b1 and b2. Likewise, the output of line (8) is an IF statement with argument A which has the same set of values before transformation (spreadsheet col. 1) and after transformation (spreadsheet col. 8) regardless of the value of A or how A is transformed to a1 and a2; the output of line (9) is an IF statement with argument B which has the same set of values before transformation (spreadsheet col. 1) and after transformation (spreadsheet col. 5) regardless of the value of B or how B is transformed to b1 and b2; and the output of line (10) is an IF statement with argument C which has the same set of values before transformation (spreadsheet col. 1) and after transformation (spreadsheet col. 6) regardless of the value of C or how C is transformed to c1 and c2.

The guard techniques recited in claims 67-69, 71, 72, 78 and 80-100 of the present application fail to satisfy Collberg's definition for an obfuscating transformation because a software program with guarding as recited in these claims will only have the same observable behavior if the run time value of the guarding variable is the same as the expected value of the guarding variable. Using language similar to Collberg, for an original program S that has a guarding variable GV (GV can be a silent guard variable, program variable, etc.) added as recited in these claims, call the guarded program S^G:

- If S fails to terminate or terminates with an error condition, then S^G may or may not terminate;
- If S terminates, and the runtime value of GV equals the expected value of GV, then S^G terminates and produces the same output as S;
- <u>However, if S terminates, and the runtime value of GV does not equal the expected value of GV, then S^G may or may not terminate and S^G will not produce the same output as S</u>.

This dependence on the runtime value of GV equaling the expected value of GV violates the basic definition of Collberg for a obfuscating transformation, and distinguishes the present invention as recited in claims 67-69, 71, 72, 78 and 80-100 over the split variable data obfuscation transformation of Collberg.

In addition to the distinguishing features discussed above, Section 7.1.3 of Collberg states:

> To allow a variable *V* of type *T* to be split into two variables *p* and *q* of type *U* requires us to provide three pieces of information: (1) a function *f(p, q)* that maps the values of *p* and *q* into the corresponding value of *V*; (2) a function *g(V)* that maps the value of *V* into the corresponding values of *p* and *q*; and (3) new operations (corresponding to the primitive operations on values of type *T*) cast in terms of operations on *p* and *q*.

(page 18, col. 2, lines 5-12). The original program in Collberg with variable V is mapped into a new program with the split variables p and q which produces the same outputs as the original program. In contrast, the guarding techniques recited in claims 67-69, 71, 72, 78 and 80-100 do not have mapping functions or new operations as required by Collberg split variables, and the guarded program does not produce the same outputs as the original program if the runtime value of the guarding variable is not equal to the expected value of the guarding variable. This also patentably distinguishes the present invention as claimed over the split variable data obfuscation transformation of Collberg.

For at least these reasons, Applicants request that the Examiner find claims 67-69, 71, 72, 78 and 80-100 patentably distinguish over Collberg, and allow claims 67-69, 71, 72, 78 and 80-100. These and other patentably distinguishing features will be discussed below with regard to the specific claims.

Claim 67

Claim 67 recites the steps of "revising the selected computation to be dependent on the runtime value of the silent guard variable, such that the selected computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable."

Collberg requires that if a program P terminates without an error condition, then the transformed program P' must terminate and produce the same output as P. (see Collberg, Section 4, page 6, col. 1 (definition of obfuscating transformation)). In contrast, the method of claim 67 recites that "the selected computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." Thus, in the method of claim 67, even if the original program terminates without an error condition, the program with silent guarding will evaluate improperly "if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable," which can cause various termination and output differences between the original program and the program with silent guarding. This violates the definition of obfuscating transformation given in Collberg and therefore claim 67 can not be anticipated by Collberg.

In addition, the split variable obfuscation of Collberg also requires mapping functions and operations that work for all values of the split variables (see Section 7.1.3, page 18, col. 2, lines 5-12; and Figure 18(a)-(e), page 19 and attached spreadsheet). The obfuscated program in Collberg performs equivalent operations to the original program. In contrast, claim 67 recites that "the selected computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." The guarded program according to claim 67 does not perform equivalent operations to the original program "if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." Thus, claim 67 also violates the split variable obfuscation requirements of Collberg. Collberg does not teach, disclose or suggest a computation that "will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable."

In the Examiner's rejection, the Examiner associates the variable "x" of Collberg, Fig. 18(e) with the silent guard variable (Office Action, page 5, section 14.a), and associates the steps of "determining an expected value of the silent guard variable at the execution point of the selected computation; and revising the selected computation . . . such that the selected computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable" with Collberg Fig. 18(e) steps 5', 7' and 8' (Office Action, page 5, section 14.c and 14.d). In addition to the above distinctions, Collberg does not teach determining an expected value for the variable "x." In Collberg, an expected value is unnecessary because the functions and operations relating A, B and C with a1, a2, b1, b2, c1 and c2 are mapping functions that work for all values of the variables (see Collberg, Fig. 18(a)-(d); page 18, col. 2, lines 5-12; and spreadsheet attached hereto). The variable "x" of Collberg is simply an intermediate variable in the data splitting transformation from A, B and C to a1, a2, b1, b2, c1 and c2, and there is no need to determine an expected value for "x" nor does Collberg teach or disclose such a step. Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 67.

For at least the reasons given above, Applicants respectfully request that the Examiner withdraw the rejection and find claim 67 to be allowable over Collberg. Claims 68-72 are dependent on base claim 67. Accordingly, Applicants respectfully request that the Examiner find claims 67-72 allowable.

Claim 69

Claim 69 is dependent on claim 68 and base claim 67 and recites the further steps of "computing the runtime value of the silent guard variable using a mathematical expression including the runtime value of the selected program variable and the expected value of the selected program variable at the dependency point." In the rejection of claim 69, the Examiner points to Collberg, Fig. 18(e) steps 5', 7' and 8' (Office Action, page 6, section 16). However, all of these steps show only the use of the runtime variable values. Collberg has no teaching of determining expected values and using both the runtime value and the expected value of a variable in a mathematical expression as recited in claim 69. Collberg teaches an obfuscating transformation that produces code that performs equivalent operations to the original code, but there is no teaching of using a mathematical expression including the runtime value of a program variable and the expected value of the program variable at a dependency point. Collberg does not disclose, teach or suggest "using a mathematical expression including the runtime value of the selected program variable and the expected value of the selected program variable at the dependency point" as recited in claim 69.

For at least this reason in addition to the reasons given with regard to claim 67, Applicants believe claim 69 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 69.

Claim 72

Claim 72 is dependent on base claim 67 and recites the further step of "inserting a mathematical expression including the runtime value of the silent guard variable and the expected value of the silent guard variable into the selected computation." In the rejection of claim 72, the Examiner points to Collberg, Fig. 18(e) steps 5'-10' (Office Action, page 7, section 18), and in the rejection of base claim 67 the Examiner associates variable "x" with the silent guard variable (Office Action, page 5, section 14.a). However, all of these steps show only the use of the runtime variable values, and no teaching or disclosure concerning an expected value, especially not for "x" which is simply an intermediate value used with the split variables. Collberg does not disclose, teach or suggest "inserting a mathematical expression including the runtime value of the silent guard variable and the expected value of the silent guard variable into the selected computation" as recited in claim 72.

For at least this reason in addition to the reasons given with regard to claim 67, Applicants believe claim 72 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 72.

Claim 78

Claim 78 recites the steps of "determining an expected value of the program variable at the point of execution of the selected computation; and revising the selected computation to be dependent on the runtime value of the program variable, such that the selected computation will evaluate incorrectly if the runtime value of the program variable is not equal to the expected value of the program variable."

Collberg requires that if a program P terminates without an error condition, then the transformed program P' must terminate and produce the same output as P. (see Collberg, Section

4, page 6, col. 1 (definition of obfuscating transformation)). In contrast, the method of claim 78 recites that "the selected computation will evaluate incorrectly if the runtime value of the program variable is not equal to the expected value of the program variable." Thus, in the method of claim 78, even if the original program terminates without an error condition, the program with guarding will evaluate "incorrectly if the runtime value of the program variable is not equal to the expected value of the program variable," which can cause various termination and output differences between the original program and the program with silent guarding. This violates the definition of obfuscating transformation given in Collberg and therefore claim 67 can not be anticipated by Collberg

In addition, the split variable obfuscation of Collberg also requires transformations and operations that work for all values of the split variables (see Section 7.1.3, page 18, col. 2, lines 5-12; and Figure 18(a)-(e), page 19 and attached spreadsheet). The obfuscated program in Collberg performs equivalent operations to the original program. In contrast, claim 78 recites that "the selected computation will evaluate incorrectly if the runtime value of the program variable is not equal to the expected value of the program variable." The guarded program according to claim 78 does not perform equivalent operations to the original program "if the runtime value of the program variable is not equal to the expected value of the program variable." Thus, claim 78 also violates the split variable obfuscation requirements of Collberg. Collberg does not teach, disclose or suggest a computation that "will evaluate incorrectly if the runtime value of the program variable is not equal to the expected value of the program variable" as recited in claim 78.

In the Examiner's rejection of claim 78, for the step of "determining an expected value of the program variable at the point of execution of the selected computation" the Examiner points to the transformation of Collberg, Fig. 18(a-e) (Office Action, page 10, section 24.g). However, the transformations shown and described in Collberg having nothing to do with expected values, but are rather functions that map variables A, B and C into split variables [a1, a2], [b1, b2] and [c1, c2], respectively. These mappings of Collberg are functions that work regardless of any expected values (see Collberg, page 18, col. 2, lines 5-12). Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 78. Collberg does not teach "determining an expected value of the program variable at the point of execution of the selected computation" as recited in claim 78.

For at least the reasons given above, Applicants respectfully request that the Examiner find claim 78 to be allowable over Collberg. Claims 79-81 are dependent on base claim 78. Accordingly, Applicants respectfully request that the Examiner find claims 78-81 allowable.

Claim 81

Claim 81 is dependent on base claim 78 and recites the further steps of "inserting a mathematical expression including the runtime value of the program variable and the expected value of the program variable into the selected computation." In the rejection of claim 81, the Examiner points to Collberg, Fig. 18(e) steps 8'-10' (Office Action, page 10, section 26). However, all of these steps show only the use of the runtime variable values. Collberg does not disclose, teach or suggest "inserting a mathematical expression including the runtime value of the program variable and the expected value of the program variable into the selected computation" as recited in claim 81.

For at least this reason in addition to the reasons given with regard to claim 78, Applicants believe claim 81 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 81.

Claim 82

Claim 82 recites the steps of "determining the expected value of the silent guard at the start of execution of the program block; and installing a branch instruction dependent on the silent guard in the software program, such that if the runtime value of the silent guard is not equal to the expected value of the silent guard then the branch instruction will cause an incorrect branch to be taken."

Collberg requires that if a program P terminates without an error condition, then the transformed program P' must terminate and produce the same output as P. (see Collberg, Section 4, page 6, col. 1 (definition of obfuscating transformation)). In contrast, the method of claim 82 recites that "if the runtime value of the silent guard is not equal to the expected value of the silent guard then the branch instruction will cause an incorrect branch to be taken." Thus, in the method of claim 82, even if the original program terminates without an error condition, the program with guarding will "cause an incorrect branch" of the branch instruction to be taken "if the runtime value of the silent guard is not equal to the expected value of the silent guard." Taking of an incorrect branch of a branch instruction can cause various termination and output differences between the original program and the program with silent guarding. This violates the definition of obfuscating transformation given in Collberg and therefore claim 82 can not be anticipated by Collberg.

In addition, the split variable obfuscation of Collberg also requires transformations and operations that work for all values of the split variables (see Section 7.1.3, page 18, col. 2, lines 5-12; and Figure 18(a)-(e), page 19 and attached spreadsheet). The obfuscated program in Collberg performs equivalent operations to the original program. In contrast, claim 82 recites that "if the runtime value of the silent guard is not equal to the expected value of the silent guard then the branch instruction will cause an incorrect branch to be taken." The guarded program according to claim 82 does not perform equivalent operations to the original program "if the runtime value of the silent guard is not equal to the expected value of the silent guard." Thus, claim 82 also violates the split variable obfuscation requirements of Collberg when the runtime value of the silent guard is not equal to the expected value of the silent guard. Collberg does not teach, disclose or suggest causing "an incorrect branch" of a branch instruction to be taken "if the runtime value of the silent guard is not equal to the expected value of the silent guard" as recited in claim 82.

In the Examiner's rejection of claim 82, for the step of "determining the expected value of the silent guard at the start of execution of the program block" the Examiner points to the transformations of Collberg, Fig. 18(a-d) and steps 1'-8' of Fig. 18(e) (Office Action, page 11, section 27.j). However, the transformations shown and described in Collberg have nothing to do with expected values, but are rather functions that map variables A, B and C into split variables [a1, a2], [b1, b2] and [c1, c2], respectively. These mappings of Collberg are functions that work regardless of any expected values (see Collberg, page 18, col. 2, lines 5-12). Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 82. Collberg does not teach "determining the expected value of the silent guard at the start of execution of the program block" as recited in claim 82.

In addition, in the Examiner's rejection of claim 82, for the step of "installing a branch instruction dependent on the silent guard in the software program, such that if the runtime value of the silent guard is not equal to the expected value of the silent guard then the branch instruction will cause an incorrect branch to be taken" the Examiner points to Collberg, page 18, section 7.1.3, especially 6th paragraph; and page 19, Fig. 18(e) steps 8'-10' after transformation. Steps 8' to 10' of Collberg Fig. 18(e) show the transformation of the IF-statements in steps 8-10 into equivalent IF-statements using the split variables. Collberg does not teach installing a branch instruction but simply translating an already existing branch instruction using split data variables. In addition, Collberg does not disclose or teach determining and using expected values of variables but rather teaches transforming variables using transformation functions as shown in Fig. 18(a-e) and page 18, col. 2, lines 5-12. Collberg does not disclose "installing a branch instruction

. . . in the software program, such that if the runtime value of the silent guard is not equal to the expected value of the silent guard then the branch instruction will cause an incorrect branch to be taken" as recited in claim 82.

For at least the reasons given above, Applicants respectfully request that the Examiner find claim 82 to be allowable over Collberg. Claims 83-87 are dependent on base claim 82. Accordingly, Applicants respectfully request that the Examiner find claims 82-87 allowable.

Claim 87

Claim 87 is dependent on base claim 82 and intervening claims 85 and 86. Claim 87 recites the further limitation that "the silent guard computation uses both the expected value of the program variable and the runtime value of the program variable." In the rejection of claim 87, the Examiner points to Collberg, Fig. 18(e) steps 8'-10' (Office Action, page 13, section 32). However, all of these steps show only the use of the runtime variable values. Collberg does not disclose, teach or suggest a computation that "uses both the expected value of the program variable and the runtime value of the program variable" as recited in claim 87.

For at least this reason in addition to the reasons given with regard to claim 82, Applicants believe claim 87 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 87.

Claim 88

Claim 88 recites "A recordable computer media having a tamper resistant software program recorded thereon, comprising: . . . a program computation . . . a silent guard variable . . . wherein the program computation is dependent on the runtime value of the silent guard variable, such that the program computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable."

Collberg requires that if a program P terminates without an error condition, then the transformed program P' must terminate and produce the same output as P. (see Collberg, Section

4, page 6, col. 1 (definition of obfuscating transformation)). In contrast, claim 88 recites that "the program computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." Thus, according to claim 88, even if the original program terminates without an error condition, a computation in the tamper resistant program will "evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." A computation evaluating improperly can cause various termination and output differences between the original program and the tamper resistant program. This violates the definition of obfuscating transformation given in Collberg and therefore claim 88 can not be anticipated by Collberg.

In addition, the split variable obfuscation of Collberg also requires transformations and operations that work for all values of the split variables (see Section 7.1.3, page 18, col. 2, lines 5-12; and Figure 18(a)-(e), page 19 and attached spreadsheet). The obfuscated program in Collberg performs equivalent operations to the original program. In contrast, claim 88 recites that "the program computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." The guarded program according to claim 88 does not perform equivalent operations to the original program "if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." Thus, claim 88 also violates the split variable obfuscation requirements of Collberg when the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable. Collberg does not teach, disclose or suggest causing a program computation to "evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable" as recited in claim 88.

In the Examiner's rejection of claim 88, the Examiner points in to Collberg, page 18, section 7.1.3; and page 19, Fig. 18(a-e) and identifies "x is the runtime value of the silent guard" (Office Action, page 13, section 33). However, the transformations shown and described in

Collberg have nothing to do with expected values, but are rather functions that map variables A, B and C into split variables [a1, a2], [b1, b2] and [c1, c2], respectively. These mappings of Collberg are functions that work regardless of any expected values (see Collberg, page 18, col. 2, lines 5-12). Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 88. Claim 88 recites "a silent guard variable . . . having an expected value at the execution point; . . . such that the program computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." The silent guard variable of claim 88 has both a runtime value and an expected value and if they are not equal "the program computation will evaluate improperly." The Examiner indicates that "x is the runtime value of the silent guard" (Office Action, page 13, section 33, lines 9-10). However, there is no indication of an expected value for "x". Collberg does not disclose the use of expected values but rather teaches mapping functions that work regardless of expected values. Collberg does not disclose, teach or suggest "a silent guard variable . . . having an expected value at the execution point; . . . such that the program computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable" as recited in claim 88.

For at least the reasons given above, Applicants respectfully request that the Examiner find claim 88 to be allowable over Collberg. Claims 89-90 are dependent on base claim 88. Accordingly, Applicants respectfully request that the Examiner find claims 88-90 allowable.

Claim 90

Claim 90, is dependent on intervening claim 89 and base claim 88. Claim 89 recites the limitation of "a program variable in the software program having an expected value at a dependency point . . . wherein the runtime value of the silent guard variable is dependent on the runtime value of the program variable at the dependency point," and claim 90 recites the further limitation of "the runtime value of the silent guard variable is also dependent on the expected value of the program variable at the dependency point." Thus, claim 90 recites that the runtime value of the silent guard variable is dependent on both "the expected value of the program variable at the dependency point" (claim 90) and "the runtime value of the program variable at the dependency point" (claim 89).

In the rejection of claim 90, the Examiner points to Collberg, Fig. 18(c-d) and (e) steps 5'-8' (Office Action, page 14, section 35). However, none of these references shows the use of an expected value. Collberg has no teaching of determining expected values or of having a computation that uses both the expected value and the runtime value of a variable as recited in claim 90. Collberg teaches transformation functions and operations that map variables into split variables regardless of expected values. Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 90. Collberg does not disclose, teach or suggest that the runtime value of a silent guard variable is dependent on both "the expected value of the program variable at the dependency point" and "the runtime value of the program variable at the dependency point" as recited in claim 90.

For at least this reason in addition to the reasons given with regard to claim 88, Applicants believe claim 90 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 90.

Claim 91

Claim 91 recites:

> A recordable computer media having a tamper resistant software program recorded thereon, comprising: . . .
>
> a program variable having an expected value at a first dependency point in the software program; and

a silent guard variable having an expected value at the first dependency point;

wherein the runtime value of the program variable is dependent on the runtime value of the silent guard variable, such that the runtime value of the program variable equals the expected value of the program variable at the first dependency point if the runtime value of the silent guard variable equals the expected value of the silent guard variable at the first dependency point.

In the rejection of claim 91, the Examiner points to Collberg, page 18, section 7.1.3, especially the 6th paragraph; page 19, Fig. 18(a-e) and states that "A, B and C are translated into a1, a2, b1, b2, c1 and c2, and x is the runtime value of the silent guard." (Office Action, page 14, section 36.n). However, none of these references in Collberg show the use of expected values for program variables and silent guard variables at a dependency point. Claim 91 recites "a program variable having an expected value at a first dependency point . . . ; and a silent guard variable having an expected value at the first dependency point." The Examiner associates "x" in Collberg, Fig. 18(e) with "the runtime value of the silent guard" but does not identify any disclosure in Collberg for the expected values of the silent guard variable or the program variable. Collberg only discloses the use of the runtime variable values, and no teaching or disclosure concerning an expected value, especially not for "x" which is simply an intermediate value used with the split variables. In Collberg, an expected value is unnecessary because the functions and operations translating A, B and C into a1, a2, b1, b2, c1 and c2 are mapping functions that work for all values of the variables (see Collberg, Fig. 18(a)-(d); page 18, col. 2, lines 5-12; and spreadsheet attached hereto). Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 91. Collberg does not disclose, teach or suggest "a program variable having an expected value at a first dependency point . . . ; and a silent guard variable having an expected value at the first dependency point; wherein the runtime value of the program variable is dependent on the runtime value of the silent guard variable, such that the runtime value of the program variable equals the expected value of the program variable at the first dependency point if the runtime value of the silent guard variable equals the expected value of the silent guard variable at the first dependency point" as recited in claim 91.

For at least the reasons given above, Applicants respectfully request that the Examiner find claim 91 to be allowable over Collberg. Claims 92-95 are dependent on base claim 91. Accordingly, Applicants respectfully request that the Examiner find claims 91-95 allowable.

Claim 93

Claim 93 is dependent on base claim 91 and includes the further limitation "a mathematical computation that includes the runtime value of the silent guard variable . . . wherein the result of the mathematical computation is corrupted if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable at the first dependency point." Collberg requires that if a program P terminates without an error condition, then the transformed program P' must terminate and produce the same output as P. (see Collberg, Section 4, page 6, col. 1 (definition of obfuscating transformation)). In contrast, claim 93 recites that "the mathematical computation is corrupted if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable at the first dependency point." Thus, according to claim 93, even if the original program terminates without an error condition, a computation in the tamper resistant program will "evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." A computation evaluating improperly can cause various termination and output differences between the original program and the tamper resistant program. This violates the definition of obfuscating transformation given in Collberg and therefore claim 93 can not be anticipated by Collberg.

In addition, the split variable obfuscation of Collberg also requires transformations and operations that work for all values of the split variables (see Section 7.1.3, page 18, col. 2, lines 5-12; and Figure 18(a)-(e), page 19 and attached spreadsheet). The obfuscated program in Collberg performs equivalent operations to the original program. In contrast, claim 93 recites that "the mathematical computation is corrupted if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable at the first dependency point." The guarded program according to claim 93 does not perform equivalent operations to the original program "if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable." Thus, claim 93 also violates the split variable obfuscation requirements of Collberg when the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable. Collberg does not teach, disclose or suggest causing a mathematical computation to be "corrupted if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable at the first dependency point" as recited in claim 93.

In the Examiner's rejection of claim 93, the Examiner points in to Collberg, page 19, Fig. 18(e) steps 5', 7' and 8' (Office Action, page 15, section 38). However, the transformations shown and described in Collberg have nothing to do with causing a mathematical computation to be "corrupted if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable at the first dependency point." Rather, Collberg shows mapping functions that map variables A, B and C into split variables [a1, a2], [b1, b2] and [c1, c2], respectively. These mappings of Collberg are functions that work regardless of any expected values (see Collberg, page 18, col. 2, lines 5-12). Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 93. Collberg does not disclose, teach or suggest the use of expected values in a mathematical computation that is "corrupted if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable at the first dependency point" as recited in claim 93.

For at least these reasons in addition to the reasons given with regard to claim 91, Applicants believe claim 93 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 93.

Claim 94

Claim 94 is dependent on base claim 91 and recites the further limitation of "a supplementary silent guard variable having an expected value at a second dependency point." In the rejection of claim 94, the Examiner points to each of the reassignments of the "C" variable shown in Collberg, Fig. 18(e) steps 4'-8' (Office Action, page 15, section 39). These steps do show a reassignment of the "C" variable, but they do not show "a supplementary silent guard variable" or "an expected value at a second dependency point" for a supplementary silent guard variable. Collberg has no teaching of supplementary silent guard variables, or of determining expected values. Collberg does not disclose, teach or suggest "a supplementary silent guard variable having an expected value at a second dependency point." as recited in claim 94.

For at least this reason in addition to the reasons given with regard to claim 91, Applicants believe claim 94 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 94.

Claim 96

Claim 96 claims "A recordable computer media having a tamper resistant software program recorded thereon, comprising . . . a silent guard in the software program having an expected value at the start of execution of the program block; and a branch instruction in the software program dependent on the silent guard, wherein the branch instruction will cause an incorrect branch to be taken if the runtime value of the silent guard is not equal to the expected value of the silent guard."

Collberg requires that if a program P terminates without an error condition, then the transformed program P' must terminate and produce the same output as P. (see Collberg, Section 4, page 6, col. 1 (definition of obfuscating transformation)). In contrast, the method of claim 96 recites that "the branch instruction will cause an incorrect branch to be taken if the runtime value of the silent guard is not equal to the expected value of the silent guard." Thus, in the method of claim 96, even if the original program terminates without an error condition, the branch instruction of the program with guarding will "cause an incorrect branch to be taken if the runtime value of the silent guard is not equal to the expected value of the silent guard." Taking an incorrect branch of a branch instruction can cause various termination and output differences between the original program and the program with silent guarding. This violates the definition of obfuscating transformation given in Collberg and therefore claim 96 can not be anticipated by Collberg.

In addition, the split variable obfuscation of Collberg also requires transformations and operations that work for all values of the split variables (see Section 7.1.3, page 18, col. 2, lines 5-12; and Figure 18(a)-(e), page 19 and attached spreadsheet). The obfuscated program in Collberg performs equivalent operations to the original program. In contrast, claim 96 recites that "the branch instruction will cause an incorrect branch to be taken if the runtime value of the silent guard is not equal to the expected value of the silent guard." The guarded program according to claim 96 does not perform equivalent operations to the original program "if the runtime value of the silent guard is not equal to the expected value of the silent guard." Thus, claim 96 also violates the split variable obfuscation requirements of Collberg when the runtime value of the silent guard is not equal to the expected value of the silent guard. Collberg does not teach, disclose or suggest a branch instruction that causes "an incorrect branch to be taken if the runtime value of the silent guard is not equal to the expected value of the silent guard" as recited in claim 96.

In the Examiner's rejection of claim 96, the Examiner points to Collberg, page 18, section

7.1.3, especially the 6th paragraph; page 19, Fig. 18(a-e) and the Examiner states that "A, B and C are translated into a1, a2, b1, b2, c1, c2 and x is the runtime value of the silent guard" (Office Action, pages 16-17, section 41). However, the translations shown and described in Collberg have nothing to do with expected values and silent guards, but are rather functions that map variables A, B and C into split variables [a1, a2], [b1, b2] and [c1, c2], respectively. These mappings of Collberg are functions that work regardless of any expected values (see Collberg, page 18, col. 2, lines 5-12). Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 96. Collberg does not teach "a silent guard . . . having an expected value at the start of execution of the program block;. . . wherein the branch instruction will cause an incorrect branch to be taken if the runtime value of the silent guard is not equal to the expected value of the silent guard" as recited in claim 96.

For at least the reasons given above, Applicants respectfully request that the Examiner find claim 96 to be allowable over Collberg. Claims 97-100 are dependent on base claim 96. Accordingly, Applicants respectfully request that the Examiner find claims 96-100 allowable.

Claim 99

Claim 99 is dependent on base claim 96 and intervening claim 98 and includes the further limitation of "a mathematical computation . . ., wherein the result of the mathematical computation is corrupted if the runtime value of the program variable is not equal to the expected value of the program variable at the insertion point." Collberg requires that if a program P terminates without an error condition, then the transformed program P' must terminate and produce the same output as P. (see Collberg, Section 4, page 6, col. 1 (definition of obfuscating transformation)). In contrast, claim 99 recites that "the mathematical computation is corrupted if the runtime value of the program variable is not equal to the expected value of the program variable at the insertion point." Thus, according to claim 99, even if the original program terminates without an error condition, a computation in the tamper resistant program will be "corrupted if the runtime value of the program variable is not equal to the expected value of the program variable at the insertion point." A computation that is corrupted can cause various termination and output differences between the original program and the tamper resistant program. This violates the definition of obfuscating transformation given in Collberg and therefore claim 99 can not be anticipated by Collberg.

In addition, the split variable obfuscation of Collberg also requires transformations and operations that work for all values of the split variables (see Section 7.1.3, page 18, col. 2, lines 5-12; and Figure 18(a)-(e), page 19 and attached spreadsheet). The obfuscated program in Collberg performs equivalent operations to the original program. In contrast, claim 99 recites that "the mathematical computation is corrupted if the runtime value of the program variable is not equal to the expected value of the program variable at the insertion point." The guarded program according to claim 99 does not perform equivalent operations to the original program "if the runtime value of the program variable is not equal to the expected value of the program variable." Thus, claim 99 also violates the split variable obfuscation requirements of Collberg when the runtime value of the program variable is not equal to the expected value of the program variable at the insertion point. Collberg does not teach, disclose or suggest causing a mathematical computation to be "corrupted if the runtime value of the program variable is not equal to the expected value of the program variable at the insertion point" as recited in claim 99.

In the Examiner's rejection of claim 99, the Examiner points in to Collberg, page 19, Fig. 18(e) steps 5', 7' and 8' (Office Action, page 17, section 44). However, the transformations shown and described in Collberg have nothing to do with causing a mathematical computation to be "corrupted if the runtime value of the program variable is not equal to the expected value of the program variable at the insertion point." Rather, Collberg shows mapping functions that map variables A, B and C into split variables [a1, a2], [b1, b2] and [c1, c2], respectively. These mappings of Collberg are functions that work regardless of any expected values (see Collberg, page 18, col. 2, lines 5-12). Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 99. Collberg does not disclose, teach or suggest the use of expected values in a mathematical computation that is "corrupted if the runtime value of the program variable is not equal to the expected value of the program variable at the insertion point" as recited in claim 99.

For at least these reasons in addition to the reasons given with regard to claim 96, Applicants believe claim 99 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 99.

Claim 100

Claim 100, is dependent on intervening claim 98 and base claim 96. Claim 98 recites the limitation of "a program variable having an expected value at an insertion point; wherein the runtime value of the silent guard is dependent on the runtime value of the program variable, such that the runtime value of the silent guard equals the expected value of the silent guard if the runtime value of the program variable equals the expected value of the program variable at the insertion point," and claim 100 recites the further limitation of "the runtime value of the silent guard is also dependent on the expected value of the program variable at the insertion point." Thus, claim 100 requires that the runtime value of the silent guard variable is dependent on both "the expected value of the program variable at the insertion point" (claim 100) and "the runtime value of the program variable at the insertion point" (claim 98).

In the rejection of claim 100, the Examiner points to Collberg, Fig. 18(e) steps 5', 7' and 8' (Office Action, page 18, section 45). However, none of these references shows the use of an expected value. Collberg has no teaching of determining expected values or of having a computation that uses both the expected value and the runtime value of a variable as recited in claim 100. Collberg teaches mapping functions and operations that map variables into split variables regardless of expected values. Each of the new instructions in Collberg (Fig. 18(e), steps 1'-10') perform an equivalent operation to the corresponding original instruction (Fig. 18(e), steps 1-10). There is no added dependence on an expected value as recited in claim 100. Collberg does not disclose, teach or suggest that the runtime value of a silent guard is dependent on both "the runtime value of the program variable" and "the expected value of the program variable at the insertion point" as recited in claim 100.

For at least this reason in addition to the reasons given with regard to claim 96, Applicants believe claim 100 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 100.

Claim Rejections – 35 USC § 103(a)

Claims 70 and 79 were rejected under 35 USC § 103(a) as being unpatentable over Collberg.

Claim 70

Claim 70, is dependent on base claim 67 and recites the additional steps of "selecting a constant value used in the selected computation; and replacing the constant value with a mathematical expression that is dependent on the runtime value of the silent guard variable, such that the mathematical expression evaluates to the constant value if the runtime value of the silent guard variable is equal to the expected value of the silent guard variable program variable."

In the rejection of claim 70, the Examiner points to Collberg, pages 18-19, section 7.1.4 and Fig. 19 (Office Action, page 19 section 49). However, section 7.1.4 and Fig. 19 of Collberg describe a method of obfuscating character strings and show that "A simple way of obfuscating a static string is to convert it into a program that produces the string." (Collberg, page 18, col. 2, ast 3 lines). Fig. 19 shows an example function G that "was constructed to obfuscate the strings

'AAA', 'BAAAA', and 'CCB'." (Collberg, page 19, col. 1, lines 4-5). The Examiner asserts that, in view of this disclosure in Collberg, the method recited in claim 70 would be obvious to one of ordinary skill in the art to at the time the invention was made. (Office Action, page 19, section 49). Applicants respectfully disagree.

In contrast to the character string obfuscation of Collberg section 7.1.4 and Fig. 19, claim 70 recites "a constant value used in the selected computation" and ""replacing the constant value with a mathematical expression." The function G of Collberg is not a mathematical expression that can be placed in a computation as recited in claim 70.

In addition, claim 70 recites that "the mathematical expression evaluates to the constant value if the runtime value of the silent guard variable is equal to the expected value of the silent guard variable." Collberg teaches transformation functions and operations that map variables into split variables regardless of expected values. By definition, the new code in Collberg using the split variables produces the same outputs as the original code. Collberg does not disclose, teach or suggest adding a dependency on an expected value of a variable as recited in claim 70.

For at least these reasons in addition to the reasons given with regard to claim 67, Applicants believe claim 70 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 70.

<u>*Claim 79*</u>

Claim 79, is dependent on base claim 78 and recites the additional steps of "selecting a constant value used in the selected computation; and replacing the constant value with a mathematical expression that is dependent on the runtime value of the program variable, such that the mathematical expression evaluates to the constant value if the runtime value of the program variable is equal to the expected value of the program variable."

In the rejection of claim 79, the Examiner points to Collberg, pages 18-19, section 7.1.4 and Fig. 19 (Office Action, page 19 section 49). However, section 7.1.4 and Fig. 19 of Collberg describe a method of obfuscating character strings and show that "A simple way of obfuscating a static string is to convert it into a program that produces the string." (Collberg, page 18, col. 2, ast 3 lines). Fig. 19 shows an example function G that "was constructed to obfuscate the strings 'AAA', 'BAAAA', and 'CCB'." (Collberg, page 19, col. 1, lines 4-5). The Examiner asserts that, in view of this disclosure in Collberg, the method recited in claim 70 would be obvious to one of ordinary skill in the art to at the time the invention was made. (Office Action, pages 19-20, section 50). Applicants respectfully disagree.

In contrast to the character string obfuscation of Collberg section 7.1.4 and Fig. 19, claim 79 recites "a constant value used in the selected computation" and ""replacing the constant value with a mathematical expression." The function G of Collberg is not a mathematical expression that can be placed in a computation as recited in claim 79.

In addition, claim 79 recites that "the mathematical expression evaluates to the constant value if the runtime value of the program variable is equal to the expected value of the program variable." Collberg teaches transformation functions and operations that map variables into split variables regardless of expected values. By definition, the new code in Collberg using the split variables produces the same outputs as the original code. Collberg does not disclose, teach or suggest adding a dependency on an expected value of a variable as recited in claim 79.

For at least these reasons in addition to the reasons given with regard to claim 78, Applicants believe claim 79 is allowable over Collberg. Accordingly, Applicants respectfully request that the Examiner so find and allow claim 79.

Final Remarks

Claims 67-72 and 78-100 are currently pending in this application and are believed to be in condition for allowance. Such allowance is respectfully requested.

In the event that there are any questions related to these amendments or to the application in general, the undersigned would appreciate the opportunity to address those questions directly in a telephone interview at 919-861-5092 to expedite the prosecution of this application for all concerned. If necessary, please consider this a Petition for Extension of Time to affect a timely response. Please charge any additional fees or credits to the account of Bose McKinney & Evans, LLP Deposit Account No. 02-3223.

Respectfully submitted,
BOSE McKINNEY & EVANS LLP

James Coles

James A. Coles
Reg. No. 28,291

Indianapolis, Indiana
(317) 684-5000

677013_2

Collberg, Fig 18e, line (5)

A	B	C=A&B
TRUE	TRUE	**TRUE**
TRUE	TRUE	**TRUE**
TRUE	FALSE	**FALSE**
TRUE	FALSE	**FALSE**
FALSE	TRUE	**FALSE**
FALSE	TRUE	**FALSE**
FALSE	FALSE	**FALSE**
FALSE	FALSE	**FALSE**

Collberg, Fig 18e, line (5')

A	a1	a2	B	b1	b2	2*a1+a2	2*b1+b2	x	c1	c2	C
TRUE	0	1	TRUE	0	1	1	1	1	0	1	**TRUE**
	1	0		1	0	2	2	1	0	1	**TRUE**
TRUE	0	1	FALSE	0	0	1	0	0	0	0	**FALSE**
	1	0		1	1	2	3	0	0	0	**FALSE**
FALSE	0	0	TRUE	0	1	0	1	3	1	1	**FALSE**
	1	1		1	0	3	2	3	1	1	**FALSE**
FALSE	0	0	FALSE	0	0	0	0	3	1	1	**FALSE**
	1	1		1	1	3	3	3	1	1	**FALSE**

Collberg, Fig 18e, line (6)

A	B	C=A&B
TRUE	TRUE	**TRUE**
TRUE	TRUE	**TRUE**
TRUE	FALSE	**FALSE**
TRUE	FALSE	**FALSE**
FALSE	TRUE	**FALSE**
FALSE	TRUE	**FALSE**
FALSE	FALSE	**FALSE**
FALSE	FALSE	**FALSE**

Collberg, Fig 18e, line (6')

A	a1	a2	B	b1	b2	a1 ^ a2	b1 ^ b2		c1	c2	C
TRUE	0	1	TRUE	0	1	1	1		1	0	**TRUE**
	1	0		1	0	1	1		1	0	**TRUE**
TRUE	0	1	FALSE	0	0	1	0		0	0	**FALSE**
	1	0		1	1	1	0		0	0	**FALSE**
FALSE	0	0	TRUE	0	1	0	1		0	0	**FALSE**
	1	1		1	0	0	1		0	0	**FALSE**
FALSE	0	0	FALSE	0	0	0	0		0	0	**FALSE**
	1	1		1	1	0	0		0	0	**FALSE**

Collberg, Fig 18e, line (7)

A	B	C=A|B
TRUE	TRUE	**TRUE**
TRUE	TRUE	**TRUE**
TRUE	FALSE	**TRUE**
TRUE	FALSE	**TRUE**
FALSE	TRUE	**TRUE**
FALSE	TRUE	**TRUE**
FALSE	FALSE	**FALSE**
FALSE	FALSE	**FALSE**

Collberg, Fig 18e, line (7')

A	a1	a2	B	b1	b2	2*a1+a2	2*b1+b2	x	c1	c2	C
TRUE	0	1	TRUE	0	1	1	1	1	0	1	**TRUE**
	1	0		1	0	2	2	1	0	1	**TRUE**
TRUE	0	1	FALSE	0	0	1	0	1	0	1	**TRUE**
	1	0		1	1	2	3	2	1	0	**TRUE**
FALSE	0	0	TRUE	0	1	0	1	1	0	1	**TRUE**
	1	1		1	0	3	2	1	0	1	**TRUE**
FALSE	0	0	FALSE	0	0	0	0	3	1	1	**FALSE**
	1	1		1	1	3	3	0	0	0	**FALSE**

Collberg, Fig 18e, line (8)	Collberg, Fig 18e, line (8')						
A	A	a1	a2	x=2*a1+a2	x==1	x==2	((x==1) || (x==2))
TRUE	TRUE	0	1	1	TRUE	FALSE	**TRUE**
TRUE		1	0	2	FALSE	TRUE	**TRUE**
FALSE	FALSE	0	0	0	FALSE	FALSE	**FALSE**
FALSE		1	1	3	FALSE	FALSE	**FALSE**

Collberg, Fig 18e, line (9)	Collberg, Fig 18e, line (9')			
B	B	b1	b2	b1 ^ b2
TRUE	TRUE	0	1	**TRUE**
TRUE		1	0	**TRUE**
FALSE	FALSE	0	0	**FALSE**
FALSE		1	1	**FALSE**

Collberg, Fig 18e, line (10)	Collberg, Fig 18e, line (10')				
C	C	c1	c2	VAL(c1,c2)	
TRUE	TRUE	0	1	**1**	**TRUE**
TRUE		1	0	**1**	**TRUE**
FALSE	FALSE	0	0	**0**	**FALSE**
FALSE		1	1	**0**	**FALSE**